<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:00 am to 8:30 am</td>
<td>Continental Breakfast</td>
</tr>
<tr>
<td>8:30 am to 8:45 am</td>
<td><em>Ptolemy Miniconferences</em>, Edward Lee (Berkeley)</td>
</tr>
<tr>
<td>8:45 am to 9:10 am</td>
<td><em>Distributed Execution Architectures in Kepler</em>&lt;br&gt;Jianwu Wang, Daniel Crawl, Ilkay Altintas, Chad Berkley, &amp; Matthew B. Jones (San Diego Supercomputer Center and UC Santa Barbara)</td>
</tr>
<tr>
<td>9:10 am to 9:35 am</td>
<td><em>Modeling Distributed Real-Time Systems with Ptolemy II</em>,&lt;br&gt;Patricia Derler, Jia Zou, Slobodan Matic, John Eidson (Berkeley)</td>
</tr>
<tr>
<td>9:35 am to 9:55 am</td>
<td><em>Semantics of Modal Models in Ptolemy</em>,&lt;br&gt;Stavros Tripakis &amp; Edward A. Lee (Berkeley)</td>
</tr>
<tr>
<td>9:55 am to 10:15 am</td>
<td>Break</td>
</tr>
<tr>
<td>10:15 am to 11:10 am</td>
<td><em>Static Analysis using the Ptolemy II Ontologies Package</em>,&lt;br&gt;Charles Shelton, Elizabeth Latronico, &amp; Ben Lickly (Bosch &amp; Berkeley)</td>
</tr>
<tr>
<td>11:10 am to 11:35 am</td>
<td><em>To Meet or Not to Meet the Deadline</em>,&lt;br&gt;Jan Reineke, Isaac Liu, Gage Eads, Stephen Edwards, Sungjun Kim, Hiren Patel (Berkeley, Columbia, Waterloo)</td>
</tr>
<tr>
<td>11:35 am to 12:15 pm</td>
<td><em>Poster Tweets</em></td>
</tr>
<tr>
<td>12:15 pm to 2:30 pm</td>
<td>Working Lunch and Poster Session</td>
</tr>
<tr>
<td>2:30 pm to 2:55 pm</td>
<td><em>The Dataflow Interchange Format: Towards Co-Design of DSP-oriented Dataflow Models and Transformations</em>,&lt;br&gt;Shuvra S. Bhattacharyya (Univ. of Maryland)</td>
</tr>
<tr>
<td>2:55 pm to 3:20 pm</td>
<td><em>Workflow Recovery for Different Models of Computation and Models of Provenance</em>,&lt;br&gt;Sven Koehler, Bertram Ludaescher, Timothy McPhillips, Anandarup Sarkar (UC Davis)</td>
</tr>
<tr>
<td>3:20 pm to 3:45 pm</td>
<td><em>Design, Analysis, and Implementation of Static Dataflow Models for Hardware Targets</em>,&lt;br&gt;Kaushik Ravindran, Murali Parthasarathy et. al, (National Instruments)</td>
</tr>
<tr>
<td>3:45 pm to 4:00 pm</td>
<td>Break</td>
</tr>
<tr>
<td>4:00 pm to 4:25 pm</td>
<td><em>Kepler/G-Pack: A Kepler Package Using the Google Cloud for Interactive Scientific Workflows</em>,&lt;br&gt;Gongjing Cao, Lei Dou, Quinn Hart, Bertram Ludaescher, (UC Davis)</td>
</tr>
<tr>
<td>4:25 pm to 4:50 pm</td>
<td><em>Context Aware Actors</em>,&lt;br&gt;Anne H.H. Ngu &amp; George Chin Jr. (Texas State Univ. &amp; Pacific NW National Lab)</td>
</tr>
<tr>
<td>4:50 pm to 5:15 pm</td>
<td><em>Modular Code Generation</em>,&lt;br&gt;Dai Bui &amp; Stavros Tripakis (Berkeley)</td>
</tr>
<tr>
<td>5:15 pm to 5:30 pm</td>
<td><em>The Ptolemy Project: Advancing System Design</em>,&lt;br&gt;Edward A. Lee (Berkeley)</td>
</tr>
<tr>
<td>6:00 pm to 8:00 pm</td>
<td>Reception and Dinner, The Faculty Club</td>
</tr>
<tr>
<td>Posters</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Gage Eads</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Shanna-Shaye Forbes</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Soheil Ghiasi, Matin Hashemi</td>
<td>UC Davis</td>
</tr>
<tr>
<td>Isaac Liu, Jan Reineke</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Slobodan Matic, Ilge Akkaya, and John Eidson</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Christian Motika, Hauke Fuhrmann, Miro Spönenmann Reinhard von Hanxleden</td>
<td>Kiel University</td>
</tr>
<tr>
<td>Chris Shaver</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Chris Shaver, Dai Bui, Stavros Tripakis</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Elizabeth Latronico, Charles Shelton, Ben Lickly</td>
<td>Bosch &amp; Berkeley</td>
</tr>
<tr>
<td>Ben Lickly, Charles Shelton, Elizabeth Latronico</td>
<td>Bosch &amp; Berkeley</td>
</tr>
<tr>
<td>Andreas Thuy</td>
<td>University of Paderborn</td>
</tr>
<tr>
<td>Stavros Tripakis, Marc Geilen, Maarten Wiggers</td>
<td>Berkeley, TU Eindhoven</td>
</tr>
<tr>
<td>Mike Wirthlin</td>
<td>Brigham Young University</td>
</tr>
<tr>
<td>Michael Zimmer</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Jia Zou, Jeff Jensen, Slobodan Matic</td>
<td>Berkeley</td>
</tr>
<tr>
<td>Jia Zou, Slobodan Matic, John Eidson</td>
<td>Berkeley</td>
</tr>
</tbody>
</table>
Edward A. Lee
Robert S. Pepper Distinguished Professor

Ninth Biennial Ptolemy Miniconference
February 16, 2011
Berkeley, CA, USA
The 1st Biennial Ptolemy Miniconference: 1995

**Ptolemy Project, Berkeley 3**

**System-Level Design of Signal Processing Systems**

**Ptolemy Research**
- Design complexity management
- Visual, algorithm level system design
- Formal methods for dataflow systems
- Programming language semantics
- Software and hardware synthesis
- Parallel architectures, partitioning, and scheduling

This highly multidisciplinary project addresses system-level design and implementation of signal processing systems.

**Ptolemy Project, Berkeley 4**

**Implementation of Signal Processing Systems**

**Hardware/Software Synthesis**
- Design of heterogeneous embedded systems
- Time-critical systems
- Synthesis of software from dataflow graphs
- System-level hardware design
- Compilation of hardware and software
- Codenaming of hardware and software

The design philosophy in Ptolemy is heterogeneous, allowing for effective use of specialized design tools within a general system-level design environment.
The 1st Biennial Ptolemy Miniconference: 1995

Heterogeneity in System-Level Design

Domains in Ptolemy

Ptolemy Project, Berkeley 5

Ptolemy Project, Berkeley 6
The 1st Biennial Ptolemy Miniconference: 1995

Where to From Here?

- Real-time scalable computing.
- Scalable embedded systems design.
- Design migration from abstract to concrete.
- Formal methods based on partial orders.
- Hybrid systems: combining FSM with dataflow.
- Modeling and analysis of random systems.
- Design of nondeterministic systems.
- Complexity management.
- Design visualization and documentation.
- Partial evaluation and incremental compilation.
- Models for back-end signal interpretation.
- Heterogeneous scheduling.

The 2nd Biennial Ptolemy Miniconference: 1997

- Formal properties.
- Scalability.
- Scheduling.
- Partitioning.
High-Performance Scalable Computing (HPSC) modeling by Sanders, a Lockheed-Martin Company.

- Yellow: start-up latency
- Blue: normal transmission/reception
- Green: processing of data on Node
- Orange: origin of contention, one or more packets queued in the switch
- Red: propagating effect of switch contention down current data path

Applications of Ptolemy in Securities Trading
or, Playing the Markets with Ptolemy

Tom Lane
Structured Software Systems, Inc.
4th 2001

Rapid Prototyping of RADAR Signal Processing Systems using Ptolemy Classic

Ptolemy MiniConference UCB
Denis Autinier, Patrick Mayer, Hane Scholler, Xavier Waznes, THALES

CONCLUSIONS (1)

- Main functional requirements are met by the final design (12 of the 13 requirements)
- Throughput and latency requirements are almost met; expected to be met in case of full speed G4 daughter cards and/or VPIPL functions redesign
- Review of graphical Ptolemy designs seems faster and more efficient than code reviews
  - Disadvantage is parameter handling and scope.
  - Design is highly modifiable, but this is difficult to see
- Total design, validate & test time for bare beamformer was 343.5 hours, while normal development takes 481 hours: Approximately 36% faster (improvement = 1.36)

Director:
- Edward A. Lee

Staff:
- Christopher Hylands
- Susan Gardner (Chess)
- Nuala Mansard
- Mary P. Stewart
- Neil E. Turner (Chess)
- Lea Turpin (Chess)

Postdocs, Etc.:
- Joern Janneck, Postdoc
- Rowland R. Johnson, Visiting Scholar
- Keies Vissers, Visiting Industrial Fellow
- Daniel Lázaro Cuadrado, Visiting Scholar

5th 2003
Graduate Students

- J. Adam Calado
- Chris Chang
- Elaine Cheong
- Sanjeev Kohli
- Xiaojun Liu
- Eleftherios D. Matsikoudis
- Stephen Neuendorf
- James Xieh
- Yang Zhao
- Hailong Zheng
- Rachel Zhou
Foundations

Our contributions:

- Behavioral Types
- Domain Polymorphism
- Responsible Frameworks
- Hybrid Systems Semantics
- Dataflow Semantics
- Tagged Signal Model
- Starcharts and Modal Model Semantics
- Discrete-Event Semantics
- Continuous-Time Semantics

Giving structure to the notion of “models of computation”

HyVisual is a targeted tool, designed for hybrid system modeling.
6th 2005

Growth of the Cal actor language

7th 2007

ReAP breakdown

The Kepler Project
Overview, Status, and Future Directions

Matthew B. Jones
on behalf of the Kepler Project team

National Center for Ecological Analysis and Synthesis
University of California, Santa Barbara
8th 2009

Cyber-Physical Systems (CPS)
Where it is going

CPS: Orchestrating networked computational resources with physical systems.

8th 2009

PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Distributed execution under DE semantics, with "model time" and "real time" bound at sensors and actuators.
Let the show begin!
Distributed Execution Architectures in Kepler

Jianwu Wang¹, Daniel Crawl¹, Ilkay Altintas¹, Chad Berkley², Matthew B. Jones²

¹ San Diego Supercomputer Center, UCSD
² National Center for Ecological Analysis and Synthesis, UCSB

Outline

• Distributed Execution Architectures in Kepler
• Master-Slave Distributed Execution Architecture in Kepler
• MapReduce Distributed Execution Architecture in Kepler
• Comparison between the Above Two Architectures
Part I

- Distributed Execution Architectures in Kepler
  - Master-Slave Distributed Execution Architecture in Kepler
  - MapReduce Distributed Execution Architecture in Kepler
  - Comparison between Master-Slave and MapReduce Distributed Execution Architectures

Distributed Execution Requirements in Kepler

- Various requirements on distributed execution in different environments, examples:
  - Ad-hoc network resources
  - Web service resources
  - Cluster resources
  - Grid resources
  - Cloud resources
  - ...
Distributed Execution Supports in Kepler

- Kepler integrated frameworks and libraries to support the requirements
  - Remote method invocation (RMI) for ad-hoc network resources
  - Axis Web service libraries for Web Service invocation
  - Ssh session libraries (JSch) for remote execution and job submission on clusters
  - Globus libraries for Grid computing

Three Distributed Execution Levels in Kepler

- **Workflow level**: the whole workflow can be executed in distributed environments
  - Example: Web service for Kepler workflow execution

- **Actor level**: distributed computing and data resources can be utilized in an actor
  - Example: Web service actor in Kepler

- **Sub-workflow level**: sub-workflows can be executed in distributed environments
  - Example: Master-Slave and MapReduce Distributed Execution
Advantages of Using Workflow System for Distributed Execution

- **Reuse existing workflows**
  - Easily transform workflow from centralized execution to distributed execution
- **Transparent implementation**
  - Hide diverse distributed techniques from users, such as different job schedulers
  - Just drag-and-drop, no coding is needed
- **Optimal execution**
  - (Semi-)automatically get the best execution plan
- **Provenance support**
- **Fault tolerance**

Part II

- Distributed Execution Architectures in Kepler
- **Master-Slave Distributed Execution Architecture in Kepler**
- MapReduce Distributed Execution Architecture in Kepler
- Comparison between Master-Slave and MapReduce Distributed Execution Architectures
Distributed Composite Actor

- As the role of Master, each token received by Distributed Composite Actor is distributed to a Slave node, executed, and the results returned.
Demo Workflow

Usability

- Users use the DistributedCompositeActor just like the common composite actor
- Interaction for execution environment transition
Part III

- Distributed Execution Architectures in Kepler
- Master-Slave Distributed Execution Architecture in Kepler
- MapReduce Distributed Execution Architecture in Kepler
- Comparison between Master-Slave and MapReduce Distributed Execution Architectures
MapReduce Actor in Kepler

(a) MapReduce actor. (b) Map sub-workflow in MapReduce actor. (c) Reduce sub-workflow in MapReduce actor.

MapReduce Actor Execution in Hadoop

1. Transfer input data from local FS to HDFS
2. MapReduce Actor creation on Hadoop
3. Transfer output data from HDFS to local FS

Master

Kepler GUI

Hadoop Master

Map slaves with input data blocks

Reduce slaves with output data blocks
Using MapReduce Actor for Word Count

Word count workflow in Kepler

Map sub-workflow

Reduce sub-workflow

Sub-workflow in IterateOverArray actor

Performance Experiment for Word Count

Graph showing performance comparison between Kepler Workflow in Hadoop and Java Program in Hadoop.
Part IV

- Distributed Execution Architectures in Kepler
- Master-Slave Distributed Execution Architecture in Kepler
- MapReduce Distributed Execution Architecture in Kepler
- Comparison between Master-Slave and MapReduce Distributed Execution Architectures

Commonalities

- Both have distributed data + distributed programs
- Both have master and slaves
- Both have execution engines on slaves
Main Differences

• **MapReduce**
  – Usually, all input data needs to be staged in beforehand and outputs is only accessible when the whole execution is finished
  – More suitable for large data sets, and has good scalability on clusters with numerous nodes

• **Master-Slave**
  – Inputs can be provided dynamically and get its result gradually once it is generated
  – More suitable for dynamic data distribution cases

Thanks!

• **Papers for the Above Work**

• **More Information:**
  – Distributed Execution Interest Group of Kepler: [https://dev.kepler-project.org/developers/interest-groups/distributed](https://dev.kepler-project.org/developers/interest-groups/distributed)
  – Contact: jianwu@sdscc.edu
MODELING DISTRIBUTED REAL-TIME SYSTEMS WITH PTOLEMY II

Patricia Derler, Jia Zou, Slobodan Matic, John Eidson, Edward A. Lee

University of California, Berkeley

DISTRIBUTED REAL-TIME SYSTEMS

Multiple computers, comprising of sensors and actuators, connected on a network that act and react on events to meet timing constraints.
OVERVIEW

• Challenges: How to model
  • Time
  • Network
  • Execution time
  • Execution semantics
• Address modeling challenges in PTIDES
Distributed platforms have different notions of time
Platform clocks drift
Platform clocks drift at varying rates

MODELING DISTRIBUTED SYSTEMS

Director mediates between actors
Difficult to maintain different notions of time
Hierarchies:
- Opaque composite actors
- Embedded directors maintain time

Top level: Oracle time
Every platform time is defined with respect to oracle time

Patrick Derler - Ptolemy Miniconference 2011
CLOCK SYNCHRONIZATION

- MasterSlave
- adjust clock rate:
- adjust clock value:

MODELING NETWORKS

Distributed platforms communicate via networks
Networks have latencies
e.g. CAN Bus, TTEthernet
Physical connections vs. Logical connections

Logical connections are lost

Aspect-oriented modeling

Quantity managers [Balarin03] and schedulers to simulate network latency

How is execution time computed?

Which time line to use for specifying execution time?

Parameter
WCET = 0.4

Aspect-oriented programming
Discrete-Event (DE) for simulation

DE as a application specification language which serves as a semantic basis for obtaining determinism in distributed real-time systems.

We need another time line

Logical time describe the execution semantics

New time line: logical time
PTIDES: AN APPLICATION
Programming temporally integrated distributed event systems [Zhao07]

- Discrete event model for execution
- Relates logical time to platform time whenever necessary
- Requires bounded error between platform clocks: Relies on clock synchronization
- Events are processed in time-stamped order

SUMMARY

- Distributed embedded systems
- Each distributed platform has its own notion of time
- Modeling distributed systems with different notions of time and clock drifts
- Clock synchronization
- Modeling networks
- Modeling distributed discrete event systems
- PTIDES
What are modal models?

- Modal models = hierarchical models mixing FSMs (Finite State Machines) and other models
Example: Hybrid System

Influences for this work

- Statecharts [Harel 87]
- Argos [Maraninchi 91]
- Esterel [Berry & Gonthier 92]
- Abstract state machines [Gurevich 93]
- Hybrid systems [Puri & Varaiya 94, Henzinger 99]
- Timed automata [Alur & Dill 94]
- SyncCharts [Andre 96]
- I/O Automata [Lynch 96]
- *Charts [Girault, Lee, & Lee 99]
- UML State machines
Ptolemy II goes one step further

Arbitrary combinations of FSMs with other domains

How to give meaning to modal models?

- Not always trivial:

What happens to the events produced by the discrete clock while system is at mode "irregular"?
How to give meaning to modal models?

- **Approach 1:**
  - Give a meaning to every possible combination of models:
    - Hierarchical state machines (Statecharts, UML, …)
    - Timed automata (timed models within state machines)
    - Hybrid automata (continuous models within state machines)
    - Mode automata (synchronous/reactive within state machines)
    - ...

  - Scalable?

- **Approach 2 [Ptolemy]:**
  - Modular semantics
    - semantics of composite blocks = function of semantics of sub-blocks
  - Compositionality
  - Heterogeneity
This talk

- A formal semantics for Ptolemy
  - operational semantics
    - close enough to the Java implementation to be faithful
    - but not too close (fits in a few pages)

- A formal semantics for modal models
Abstract semantics

- **Actor** = State Machine
- **Actor** = Inputs + Outputs + States + Initial state + Fire + Postfire

- Fire = output function: produces outputs given current inputs + state \( F : S \times I \rightarrow O \)
- Postfire = transition function: updates state given current inputs + state \( P : S \times I \rightarrow S \)

Implemented as Java interfaces

**Interface “Initializable”**

**Method Summary**

- void **fire()**
- boolean isFireFunctional()  Examine if this executable does not change state in either the prefire() or the fire() method.
- boolean isLocked()  Examine if this executable is strict, meaning all inputs must be known before iteration.
- int iterate(int count)  Invoke a specified number of iterations of the actor.
- boolean postfire()  The method should be invoked once per iteration, after the last invocation of fire() in that iteration.

**Interface “Executable”**

**Method Summary**

- void **fire()**
- boolean isFireFunctional()  Examine if this executable does not change state in either the prefire() or the fire() method.
- boolean isLocked()  Examine if this executable is strict, meaning all inputs must be known before iteration.
- int iterate(int count)  Invoke a specified number of iterations of the actor.
- boolean prefire()  This method should be invoked prior to each invocation of fire().
- void stop()  Request that execution of this Executable stop as soon as possible.
- void stopEvents()  Request that execution of the current iteration complete.
- void terminate()  Terminate any currently executing model with extreme prejudice.
Examples

Single state ("stateless").
P: trivial (state never changes).
F: out := in*1.5

State: the current value
F: out := state
P: state := input

Behaviors – untimed

Set of untimed traces:

\[ s_0 \xrightarrow{x_0, y_0} s_1 \xrightarrow{x_1, y_1} s_2 \xrightarrow{x_2, y_2} \cdots \]

such that for all i:

\[ y_i = F(s_i, x_i) \]
\[ s_{i+1} = P(s_i, x_i) \]
What about “timed” actors?

- States include special **timer** variables:
  - Set to some positive value, “expire” when they reach 0, can be “frozen” and “resumed”, …

Behaviors – timed

- Set of timed traces:
  
  \[
  S_0 \xrightarrow{x_0, y_0, d_0} S_1 \xrightarrow{x_1, y_1, d_1} S_2 \xrightarrow{x_2, y_2, d_2} \ldots
  \]

- such that for all \( i \):
  
  \[
  y_i = F(s_i, x_i) \\
  s_{i+1} = P(s_i - d_i, x_i) \\
  d_i \leq \min \{ v \mid v \text{ is the value of a timer in } s_i \}
  \]

\[
F : S \times I \rightarrow O \\
P : S \times I \rightarrow S
\]
What about hierarchy?

How to give semantics to a hierarchical model?  
i.e.,  
How to give semantics to a composite actor?

Directors = composition operators

- Given a composite actor with a set of subactors \( A_1, A_2, \ldots \), with fire & postfire functions \( F_1/P_1, F_2/P_2, \ldots \)

- … its director defines a new pair of fire & postfire functions \( F \) and \( P \).

- \( F \) and \( P \) define a new, composite actor \( A \).

- \( A \) can be used like an atomic actor (black-box).
Example: Synchronous/Reactive (SR)

- To compute the composite $F$, director solves a fixpoint:
  - Keep on evaluating $F_i$'s until the values of all signals stabilize (this includes output signals in particular)
  - State remains unchanged during computation of the fixpoint!
    c.f. separation of fire and postfire
- To compute the composite $P$, just execute all $P_i$'s

Example: Synchronous Data Flow (SDF)

In each firing, actors consume a fixed number of tokens from the input streams, and produce a fixed number of tokens on the output streams.

- SDF Director computes periodic schedule, e.g., $A,A,A,B,B$
- Composite fire() fires all internal actors according to schedule
MODAL MODEL SEMANTICS

Giving semantics to modal models

Goal: define $F$, $P$ functions for the modal model

$F_c$, $P_c$ functions already defined for the "controller" automaton

$F_2$, $P_2$ functions already defined for this refinement

$F_1$, $P_1$ functions already defined for this refinement
Rough description of semantics

- Given current controller state $s_i$:
  - If no outgoing transitions from $s_i$ are enabled:
    - Use $F_i$ and $P_i$ to compute $F$ and $P$
  - If preemptive outgoing transitions from $s_i$ are enabled:
    - Use the actions of these transitions to compute $F$ and $P$
  - If only non-preemptive outgoing transitions from $s_i$ are enabled:
    - First fire refinement, then transition, i.e.:
      - $F$ is the composition of $F_i$ and the output action of a transition
      - $P$ is the composition of $P_i$ and the state update action of a transition
  - Timers of refinements suspended and resumed when exiting/entering states
  - Details in paper "Modal Models in Ptolemy" [EOOLT 2010]

Timed modal model example

Mode transitions triggered at times 0, 2.5, 5, 7.5, etc.

Events with value 1 produced at (local times) 0, 1, 2, 3, etc.

First regular event generated at (global time) 0, then transition is immediately taken.

Spontaneous Modal Model
Conclusions, ongoing work and future challenges

- Modular formal semantics for Ptolemy II
  - Directors = composition operators over state machines

- Semantics worked out for modal models [EOOLT 2010]
  - Currently extending it to other domains: Synchronous-Reactive, SDF, Discrete-Event, Continuous-Time, …

- Meta-model to describe semantics?

Thank you

- Questions?
Static Analysis using the Ptolemy II Ontologies Package

Charles Shelton, Elizabeth Latronico (Bosch Research and Technology Center)  
Ben Lickly, Edward Lee (UC Berkeley)

Ptolemy Mini-Conference, February 16, 2011

Motivation

- Cars are networked software systems
  - Up to 70 Electronic Control Units
  - Software crucial for many features
    - Electronic stability control
    - Parking assist
    - Emissions control
    - Engine Start/Stop
    - Active and passive safety
  - Bosch makes all of these systems for auto manufacturers
- How can we manage increasing complexity and interconnectedness of software models?
- Analysis approaches promising, but hand-annotation has drawbacks
  - Time intensive to develop and maintain
  - People are inconsistent, make errors
  - Repeat for every composition
Examples of Model Construction Errors

- **Units error**
- **Transposition error**
- **Semantics error**

**Static Analysis Using the Ptolemy II Ontologies**

**Static Analysis Using Ontologies**

- An **Ontology** consists of:
  - A set of **Concepts**
  - **Relationships** between those concepts

- Ontologies are used for representation of semantic information
  - General ontology frameworks (e.g., OWL) focus on expressiveness
  - Arbitrary ontologies represent complex relationships as a graph

- Restrict Ptolemy ontologies to **lattice** graph structure
  - Lattice elements form a complete partial order
  - Existing scalable analysis algorithms
  - Existing work from compiler static analysis
Ontology Example: Ptolemy Type System

- Ptolemy type system implementation
  - Types organized in a lattice
  - Edges represent “can be converted to” relationships
  - Automatic type inference and propagation
    - Rehof-Mogensen constraint solving algorithm
- Users define type constraints in their actors
  - eg. An actor’s output port type must be “greater than or equal to” (higher in the lattice) the input port type
  - Connections between actors imply that the sink type $\geq$ the source type

Ptolemy Ontologies Framework

- Ontologies package generalizes the Ptolemy type system framework
  - Users can define their own ontology
  - Must also define the rules that determine ontology concept resolution
    - Constraints between model elements
    - Constraints between actor input and output ports
  - Reuses existing Ptolemy code
- Constraints are specified as inequalities between concepts assigned to each model element
  - $c_{output} \geq c_{input}$
  - $c_{output} \geq f(c_{input})$ where f is a monotonic function in the ontology domain \( (c_a \geq c_b \implies f(c_a) \geq f(c_b)) \)
Demo: Dimensional Analysis

- Use the Ontology static analysis to infer dimensional properties
  - Position, Velocity, Acceleration, Time
- Ptolemy Model Example: Simple Car Dynamics Model
  - There is an error in this model that leads to incorrect results

Step 1: Drag in a Lattice Ontology Solver
Static Analysis Using Ptolemy II Ontologies

Step 1: Drag in a Lattice Ontology Solver

Step 2: Open the Solver Model
Step 2: Open the Solver Model

Step 3: Drag an Ontology into the Solver Model
Step 3: Drag an Ontology into the Solver Model

Step 4: Create the Dimensional Analysis Ontology
Step 4: Create the Dimensional Analysis Ontology

Step 5: Add Actor Constraints to the Solver Model
Step 5: Add Actor Constraints to the Solver Model
Step 5: Add Actor Constraints to the Solver Model

Defining Actor-Specific Constraints

<table>
<thead>
<tr>
<th>Actor</th>
<th>Elements</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrator</td>
<td>input port derivative (x), output port state (y)</td>
<td>$c_y \geq f_i(c_x)$, $c_y \geq f_o(c_x)$</td>
</tr>
</tbody>
</table>

$f_i(c_y) = \begin{cases} 
\text{Unknown} & \text{if } c_y = \text{Unknown} \\
\text{Velocity} & \text{if } c_y = \text{Position} \\
\text{Acceleration} & \text{if } c_y = \text{Velocity} \\
\text{Dimensionless} & \text{if } c_y = \text{Time} \\
\text{Conflict} & \text{Otherwise} 
\end{cases}$

$f_o(c_x) = \begin{cases} 
\text{Unknown} & \text{if } c_x = \text{Unknown} \\
\text{Position} & \text{if } c_x = \text{Velocity} \\
\text{Velocity} & \text{if } c_x = \text{Acceleration} \\
\text{Time} & \text{if } c_x = \text{Dimensionless} \\
\text{Conflict} & \text{Otherwise} 
\end{cases}$
Step 5: Add Actor Constraints to the Solver Model
Step 5: Add Actor Constraints to the Solver Model

Static Analysis Using Ptolemy II Ontologies

Research and Technology Center

Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 | © 2011 Robert Bosch LLC and affiliates. All rights reserved.
Step 5: Add Actor Constraints to the Solver Model
Static Analysis Using Ptolemy II Ontologies

Step 5: Add Actor Constraints to the Solver Model

Execute the Lattice Ontology Analysis
Step 6: Add Initial Constraints to the Model
Step 6: Add Initial Constraints to the Model
Step 6: Add Initial Constraints to the Model

Reexecute the Lattice Ontology Analysis
Fix the Model Error and Reanalyze

This erroneously connects Acceleration output data to an input expecting Velocity data.
Done!

Potential Uses for Ontology-Based Analyses

- **Type/Semantics Checking**
  - Signal Data type Propagation
  - Signal Physical Dimension Propagation
  - Signal Physical/Logical Propagation
  - Signal Data/Control Propagation
- **Constant/Non-Constant Propagation**
- **Reachability**
- **Observability**
- **Identify and Propagate Diagnostic/Functional Model Elements**
Static Analysis Using the Ptolemy II Ontologies

Ongoing Work

- Combining multiple ontology frameworks for integrated analyses
  - POSTER: Elizabeth Latronico

- Ontologies with Infinite Lattice Elements
  - Constant value propagation
  - Representing and propagating records of lattice elements
  - POSTER: Ben Lickly

- Concept function monotonicity analysis
  - Automatically determine whether or not a function is monotonic
  - Enable easier development of ontology frameworks

- Ontology Error Analysis
  - Identify errors in the model by finding specific constraint conflicts

Conclusions

- Lattice-based ontologies enable automatic static analysis
  - Models can be verified for structural and semantic properties
  - Guaranteed sound analysis given:
    - The ontology is a lattice
    - All constraint functions are monotonic
  - Analysis algorithm scales with the number of constraints
    - # constraints scales with # model elements

- Ontologies Package Demos in the Ptolemy Repository
  - /ptolemy/data/ontologies/demo
    - /ptolemy/data/ontologies/demo/DimensionSystemExample
    - /ptolemy/data/ontologies/demo/CarTracking

- Thanks!
  - Charles.Shelton@us.bosch.com
Abstractions are Great

… if they abstract the right thing

Higher-level Model of Computation
C-level programming language
Instruction Set Architecture (ISA)
Hardware Realizations

Abstracts from execution time

Code Generation
Compilation
Execution

To Meet or Not to Meet the Deadline

Gage Eads  
Stephen A. Edwards  
Sungjun Kim  
Edward A. Lee  
Ben Lickly  
Isaac Liu  
Hiren D. Patel  
Jan Reineke <speaker>

Ninth Biennial Ptolemy Miniconference
Berkeley, CA, February 16, 2011
Current Timing Verification Process

- C Program
- Compiler
- Binary
- Architecture
- WCET Analysis

Current Timing Verification Process

- New Architecture → Recertification
- Extremely time-consuming and costly

Airbus: 40 years supply of
**Agenda of PRET**

- Higher-level Model of Computation
- C-level programming language
- Instruction Set Architecture (ISA)
- Hardware Realizations

**PRET Machines**

Make Timing a Semantic Property of Computers

**Precision-Timed (PRET) Machines**

Timing precision with performance: Challenges:
- Memory hierarchy (scratchpads?)
- Deep pipelines (interleaving?)
- ISAs with timing (deadline instructions?)
- Predictable memory management (Metronome?)
- Languages with timing (discrete events? Giotto?)
- Predictable concurrency (synchronous languages?)
- Composable timed components (actor-oriented?)
- Precision networks (TTA? Time synchronization?)

See our posters!
Agenda of this Talk

- Higher-level Model of Computation
- C-level programming language
- Instruction Set Architecture (ISA)
- Hardware Realizations

Adding Control over Timing to the ISA

Variant 1: “delay until”

Some possible capabilities in an ISA:

- [V1] Execute a block of code taking at least a specified time [Ip & Edwards, 2006]

Where could this be useful?

- Finishing early is not always better:
  - Scheduling Anomalies (Graham's anomalies)
  - Communication protocols may expect periodic behavior
  - …

Reineke et. al, Berkeley 7

Reineke et. al, Berkeley 8
Adding Control over Timing to the ISA
Variants 2+3: "late" and "immediate miss detection"

- **[V2]** Do [V1], and then conditionally branch if the specified **time** was exceeded.

  ![Diagram](branch_expired)

- **[V3]** Do [V1], but if the specified **time** is exceeded during execution of the block, branch immediately to an exception handler.

  ![Diagram](exception_on_expire)

Applications of Variants 2+3
"late" and "immediate miss detection"

- **[V3]** "immediate miss detection":
  - Runtime detection of missed deadlines to initiate error handling mechanisms
  - Anytime algorithms
  - However: unknown state after exception is taken

- **[V2]** "late miss detection":
  - No problems with unknown state of system
  - Change parameters of algorithm to meet future deadlines
**PRET Assembly Instructions**
Supporting these Four Capabilities

- `set_time %r, <val>`
  - loads current time + `<val>` into `%r`
- `delay_until %r`
  - stall until current time >= `%r`
- `branch_expired %r, <target>`
  - branch to target if current time > `%r`
- `exception_on_expire %r, <id>`
  - arm processor to throw exception `<id>` when current time > `%r`
- `deactivate_exception <id>`
  - disarm the processor for exception `<id>`

---

**Controlled Timing in Assembly Code**

[V1] Delay until:

```
set_time r1, 1s
// Code block
delay_until r1
```

[V2] Late miss detection

```
set_time r1, 1s
// Code block
branch_expired r1, <target>
delay_until r1
```

[V3] Immediate miss detection

```
set_time r1, 1s
exception_on_expire r1, 1
// Code block
deactivate_exception 1
delay_until r1
```

[V2] + [V3] could all have a variant that does not control the minimum execution time of the block of code, but only controls the maximum.
### Application: Timed Loops

**Fixed Period**
- `set_time r1, 1s`
- `loop:
  // Code block
  delay_until r1`
- `r1 = r1 + 1s`
- `b loop`

**Lower bound for each iteration**
- `set_time r1, 1s`
- `loop:
  // Code block
  delay_until r1`
- `set_time r1, 1s`
- `b loop`

The two loops above have different semantics:

![Diagram showing the differences between fixed period and lower bound for each iteration.]

### Timed Loop with Exception Handling

**Exact execution time (no jitter)**
- `set_time r1, 1s`
- `exception_on_expire r1, 0`
- `loop:
  // Code block`
- `deactivate_exception 0`
- `delay_until r1`
- `r1 = r1 + 1s`
- `exception_on_expire r1, 0`
- `b loop`

This code takes exactly 1 second to execute each iteration. If an iteration takes more than 1 second, then as soon as its time expires, the iteration is aborted and an exception handler is activated.
Exporting the Timed Semantics to a Low-Level Language (like C)

This realizes variant 2, “late miss detection.”

The code block will execute to completion. If 500ms have passed, then the patchup procedure will run.

Reineke et al, Berkeley 15

---

Exporting the Timed Semantics to a Low-Level Language (like C)

This pseudo-code is neither C-level nor assembly, but is meant to explain an assembly-level implementation.

Reineke et al, Berkeley 16
Variant with Exact Execution Times:

```
try for (500ms) {
    // Code block
    catch {
        panic();
    }
}
```

This is the same, except for the added `delay_until`

```
jmp_buf buf;
if (!setjmp(buf)) {
    set_time r1, 500ms
    exception_on_expire r1, 0
    // Code block
    deactivate_exception 0
    delay_until r1
} else {
    panic();
}
exception_handler_0 () {
    longjmp(buf)
}
```

MTFD – Meet the F(inal) Deadline

- Variant [V1] ensure that a block of code takes at least a given time.
- Variants [V2, V3] allow to act upon deadline misses.
- [V4] “MTFD”: Execute a block of code taking at most the specified time.

Being arbitrarily “slow” is always possible and “easy”.

But what about being “fast”?

```
set_time r1, 1s
// Code block
MTFD r1
delay_until r1
```

Reineke et. al, Berkeley 17
Current Timing Verification Process

- C Program
- Compiler
- Binary
- Architecture
- WCET Analysis

Reineke et. al, Berkeley 19

Current Timing Verification Process

- C Program
- Compiler
- Binary
- Architecture
- WCET Analysis

- New Architecture → Recertification
- Extremely time-consuming and costly

Reineke et. al, Berkeley 20
The Future (?) Timing Verification Process

- Timing is property of ISA
- Compiler can check constraints once and for all
- Downside: little flexibility in architecture development

The Future (?) Timing Verification Process: More Realistic?

- ISA leaves more freedom to implementations
- Compiler generates constraints on architecture to meet timing constraints
Conclusions

- Abstractions are great, if they are the right abstractions
- Real-time computing needs different abstractions

*Raffaello Sanzio da Urbino – The Athens School*
Deadline Instructions in a PRET Architecture

Gage Eads, Edward A. Lee, Isaac Liu, Hiren D. Patel, Jan Reineke

We design timing constructs such as lower and upper bounded execution through a combination of PRET timing instructions. A fourth construct, MTFD, would statically determine whether a code block can meet its deadline on a given predictable system.

C Level Constructs

Real-time software engineers require expressive timing constructs at the programming language level. We present three possible control blocks, constructed from PRET deadline instructions: `tryin expired()`, `tryin catch()`, and `tryfor()`. Since then, we have made significant changes to the PRET architecture: the ISA changed from SPARCv8 to ARMv4, a predictable memory controller subsumed the memory wheel, and the core became a 4-stage pipeline. Version 2.0 of the PRET Simulator is functional and features all of the aforementioned deadline instructions. It will be released for public use under an open-source license in the near future.

Assembly Timing Instructions

We augment the ARM ISA with five powerful timing instructions. The instructions are implemented as coprocessor accesses to the 64-bit timer coprocessor, which is partitioned into a 32-bit second counter and a 32-bit nanosecond counter. Register operands `rd` and `rm` contain the second and nanosecond values, respectively.

PRET Simulator

We released the PRET Simulator v1.0 in February 2009. A number of class projects and papers leveraged the simulator to explore precision timed software design, such as an automated mapping from a timed functional specification (Giotto) to a PRET architecture.

Acknowledgments

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720892 (CSR-EHS:PRET) and #0720841 (CSR-CPS)), the U.S. Army Research Office (AR01W911NF-07-2-0019), the U.S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota. The authors acknowledge the support of the Multiscale Systems Center, one of six research centers funded under the Focus Center Research Program.
The focus of this work is on timing errors. A timing error occurs when the specification says one thing and the implementation does something else. Often times, this is caused by execution at a time that violates a specification.

* Error Transitions

Methodology

Model-based design, simulation, and synthesis is being used more than before in lieu of hand writing code and testing it [4].

There is also a resurgence in Cyber Physical System design due to renewed interest in the area

If these trends continue, we will see:

1. More use of model-based design with timing specifications in the design of Cyber Physical Systems
2. The desire to include error handling explicitly in a model instead of in an ad hoc manner

Discussion and Future Work

Error Handling in Model-Based design for Real-Time Systems is still a work in progress. Preliminary results indicate that the current mechanism is able to detect and appropriately transition after simulating a timing overrun.

1. Error Transition for timing errors into modal models [3]
2. Timing manager to introduce a secondary notion of time and handle errors hierarchically
3. Design Assistant to aid the user
4. Preliminary code generation support for the timing manager

Future Work

1) Incorporating representative probabilistic distributions into the timing
3) expanding the preliminary work in C and Java code generation

Allowing the user to specify a recovery transition
Adding in other types of timing error transitions

Modeling

ARoyal Airforce pilot accidentally dropped a practice bomb on the flight deck of the Royal Navy's aircraft carrier. It missed it's intended target and several sailors were injured. The cause was attributed to a timing delay in the software.

In an effort to avoid possible similar mistakes with newly designed systems and to provide a more systematic means of dealing with timing errors, we present preliminary work that extends a model of computation (MOC) for embedded systems which features timing semantics.

More recently, the SPIRIT Mars rover encountered a "reboot loop" shortly after landing, where a fault during the booting process caused the system to reboot again. Luckily, a software patch solved the problem and the mission continued successfully [2].

A timing error is detected the timing manager passes the error up the model hierarchy. If there is not error transition to catch the error in the hierarchy an exception the simulation of the model is stopped and the user is informed of the unhandled error.

Notes desire to incorporate physical time into the model

1. Notes desire to incorporate physical time into the model
2. Annotates actors with a execution time estimate parameter. Values are expected to be provided by the designer or by an external execution time analysis tool
3. Simulates execution time of actors as a probabilistic variant of annotated execution time estimate parameter value
4. If a timing error is detected the timing manager passes the error up the model hierarchy. If the specification is included in a modal model the timing manager enables the first applicable error transition. If not it moves further up the hierarchy and attempts to enable an error transition
5. If there is not error transition to catch the error in the hierarchy an exception the simulation of the model is stopped and the user is informed of the unhandled error

References


Acknowledgements

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHES) at UC Berkeley, which receives support from the National Science Foundation (NSF) awards MII-0726932 (CSR-0726932) and MII-0728401 (CSR-0728401), the U. S. Army Research Office (ARO/DMII) W911NF-07-0-0419, the U. S. Air Force Office of Scientific Research (MUO) FA8650-06-1-3132, the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies; Agilent, Bosch, HSI, Lockheed Martin, National Instruments, and Toyota.

This work was also supported by the Jenkins Predoctoral Fellowship program as well as a Jenkins Predoctoral Fellowship Program Mini Grant
FORMLESS: Scalable Utilization of Embedded Manycores in Streaming Applications

Matin Hashemi, Soheil Ghiassi
Electrical and Computer Engineering Department
University of California, Davis
http://leps.ece.ucdavis.edu

**Throughput Estimation:**
- Theorem: Assuming large inter-processor buffers, for any task assignment of dataflow graph $G$ to $P$ processors, there exists an ordering of tasks on processors such that every precedence constraint is met (possibly by overlapping iterations), and the steady state execution period (inverse of throughput) of the application is $EP = \max_{p=1}^{P}\text{workload}(p)$.

**Task Assignment**
- Objective: minimize execution period $EP$
- Constraint: each processor should have at most 4 connections from and to other processors
- Implementation:
  - METIS graph partitioning software
  - Post partitioning adjustment to meet the above constraint

**Local Scheduling**
- Overlap execution of different iterations of the application tasks
- Must respect the dependencies
- Increases throughput
- Increases memory requirement for inter-task buffers

**Baseline Software Synthesis**
- **Design Space Exploration**
  - Throughput Estimation
  - Task Graph Generation
- **FORMLESS Application**
  - Task Assignment
  - Task Profiling
- **Instantiated Task Graph**
  - Local Scheduling
  - Task Assignment
- **SEAM Simulator**
  - Processor Assignment
  - Code Generation
- **FPGA Prototyped Platform**
  - Advanced Encryption Standard (AES)
  - Low Density Parity Check (LDPC)
  - Matrix Multiply
  - FPGA Prototyped Platform
    - Altera DE4 FPGA board
    - 16 non-isolated cores
    - Instruction cache: 8KB
    - Data cache: 32KB
    - FIFO depth: 1024
Mission

The traditional computing abstractions only concern themselves with the “functional” aspects of a program and not its timing properties. This allows the use of techniques like speculative execution, caches, interrupts, and dynamic compilation that offer improved average-case performance at the expense of predictable execution times. The PRET project aims to improve the timing predictability at all layers of abstraction by carefully reexamining and reworking various architectural and compiler advancements with an eye toward their effects on timing behavior and worst-case bounds.

Computer Architecture

In our research, we have pursued a bottom-up approach. Starting with the underlying pipeline, we have modified and added constructs to improve the timing predictability at the architectural and instruction-set-architecture level.

Interleaved multithreading

By using interleaving threads through our pipeline, we are able to remove the data hazard and dependencies in the pipeline, creating a timing predictable pipeline.

Memory Hierarchy

Conventional memory systems uses a hierarchy of memory units to bridge the latency. CPUs use registers for fast data processing operations. However, the memory hierarchy is designed as "best effort" latency and bandwidth requirements. Our goal is to look at modern memory hierarchies and design a predictable memory system.

A DRAM device consists of several banks along with controller logic to decode addresses. Concurrent accesses to banks are possible, but I/O pins are shared.

Predictable DRAM Access

DDR2 DRAM devices utilizes bank parallelism to achieve better performance. It's difficult to predict DRAM access time because accesses to the same bank need to wait for the previous access to finish, while accessing a different bank can be done concurrently. Our DRAM controller exposes groups of banks as independent resources that are accessed sequentially which hides the latency of accessing a single bank. A TDMA scheduler provides predictable access latencies.

Scratchpad Memories

Caches can use different hardware replacement policies in attempt to prefetch the data needed from main memory. However, when the software issues a load or store instruction, it does not know the state of the cache, or what data has been prefetched in it. Scratchpad memories are another form of fast access memory which is managed in software, much like registers that use explicit load/store instructions to manipulate contents for fast data processing. With software management, better real time guarantees can be provided.

Above shows one possible memory hierarchy for PRET. The memory system is a major source of headache for analyzing or predicting execution, as it causes a wide range of execution time for even the same programming running on the same system. Our goal is to design a predictable memory system that provide systems with predictable memory access times.

Acknowledgement

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS:PRET) and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO#W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, HSBC, Lockheed-Martin, National Instruments, and Toyota.
Distributed Testbed Objectives

- Safe event processing
- Bounded latency between hardware components
- Distributed bound-delay networking protocol
- Expressiveness of interaction
- Complexity of real sensor, actuator and network interfaces
- Timing support
  - External to microprocessor

- Scheduling
  - Deadlines defined by sensor – actuator model delays

- Network addressing
- Visibility during model-based design

Programming Model

- Programming Temporally Integrated Distributed Embedded Systems
- Based on Discrete-Event semantics
- Event processing in time-stamp order

Relates model time to physical time at environment interfaces

- \( d_0 \) – sensor latency

Leverages time synchronization across distributed platforms

- \( \text{(IEEE 1588 protocol over Ethernet)} \)
- \( d = \text{comm. latency}, \tau = \text{sync. error} \)

Preliminary Experiments

- \( t(\text{in}_2) - t(\text{in}_1) = 3\mu s \)
- \( t(\text{in}_2) - t(\text{out}) = 50\mu s \)

- \( t(\text{in}_2) - t(\text{out}) = 50\mu s \)

- \( t(\text{in}_2) - t(\text{out}) = 50\mu s \)

Synchrophasor-based Measurement and Control

- Frequency and phase of power generation units must remain synchronous
- Synchrophasor technology provides a powerful tool to directly measure the state
- PMU = Primary Measurement Unit
- SVP = Synchrophasor Vector Processing unit

Real PTDIeS problem

- Tight (1-5us) timing accuracy
- Reasonable sample rate (4.8 KSPS)
- Timing based on UTC
- Multiple functions run concurrently (UDP, PTP)
- Naturally distributed due to physical size of grid

Ptolemy Approximation Model
**KIELER Actor Oriented Modeling**

**Kiel Integrated Environment for Layout** (Eclipse Rich Client)

---

**Meta Modeling**

KAOM – KIELER Actor Oriented Modeling

- XMI
- Text: Xtext
- Diagrams: EMF
- Transient EMF (serializable model)
- GMF Compose
- Variability
- Subset EMF (Eclipse GMF)
- Graph

**Model Execution**

KIEEM – KIELER Execution Manager [3]

- Overview of the Execution Manager Infrastructure
- DMK/MIKI
- Reference Model
- Execution Manager Engine
- Task Manager

**View Management**

KIVI – KIELER View Management

- The Model: E.g., System, Component, Class, Method
- The View: Processing, Visualization, Simulation, Traceability

**Data Visualization**

- Visualization of a Model Instance
- Traffic Light Environment Visualization

**Automatic Layout**

KIML – KIELER Infrastructure for Meta Layout [2, 5]

- Automatic layout of GMF and Graphiti diagrams
- Generic interface for layout algorithms
- Flexible configuration of layout options by the user
- Automatic layout integrated into Ptolemy’s graphical editor Vergil

**Model Rendering**

KARMA – KIELER Advanced Rendering for Model Appearance

- Possible customized model rendering options:
  - E.g., Ptolemy, LabView, Simulink, ASCET, SCADE
  - GMF editor support, Graphiti (planned)

---

**KAOM with Automatic Layout, Ptolemy rendering and Simulation in KIELER**

---

**Contact Person:**
Prof. Dr. Reinhard von Hanxleden
Department of Computer Science
Christian-Albrechts-Universität zu Kiel
Olshausenstrasse 40, 24098 Kiel, Germany
Phone: +49 (0) 431 880-7281 / 7526
Fax: +49 (0) 431 880-7615
rvh@informatik.uni-kiel.de
http://www.informatik.uni-kiel.de/rtsys

---

**Further Information:**
http://www.informatik.uni-kiel.de/rtsys/kieler

---

*Poster presented at 9th Biennial Ptolemy Miniconference (PTCONF11), Berkeley, CA, February 2011, © Christian Motika, Miro Spörmann, Reinhard von Hanxleden 2011*
Using Ptolemy/VisualSim for Internet-based Model Sharing & Communication

Darryl Koivisto
Chief Technology Officer
dkoivisto@mirabilisdesign.com
2/14/11 Mirabilis Design

Mission Statement

System Simulation Software

with Application-specific Modeling Libraries

for Performance and Architecture Exploration

of Large Complex Electronic Systems

Innovative methodology for addressing today's Ad-Hoc Approaches

About Mirabilis Design

System design and exploration for Systems, SoC

VisualSim- Modeling and simulation environment

Based in Silicon Valley with offices in Southern California, Czech and India

Experienced application support in US, India, Japan and China

Largest source of system modeling IP with embedded timing and power

Experts in system modeling and architectures

Select the “Right” configuration to match customer request

What is a System?

Satellite

Networking

Defense

Protocols

Computer

Automotive

Semiconductors

Industrial

Electronics

Wireless

Digital Control

uP and Computing

etc...

System is what you are building

Yesterday's System

Today's System

Challenges - Convergence

Dept. A

Baseband

FFT

IIR

Dept. B

Protocol

FSM

Dept. C

Analog/RF

Dept. D

Video

C/C++

Convergence of Multimedia, Computing and Communications

Challenges - IP Re-Use

Internet reduced IP distribution cost, But …

• No standard modeling interface and format

• Several abstraction-levels and domains

• No unified design and simulation environment

It is possible...

Non-standard formats prevent IP reuse over the Internet

IP Distribution and Evaluation

WW IPware URL

My_URL_list

IP Packaging similar to Adobe Publisher

Creating the Right Design

Integrate!

To Implementation (HDL, Embedded C/C++/Java)

Peripherals

Protocol

Microcontroller

RISC CPU

I/O

Communications

Input

Output

Integrate!
Multidimensional Dataflow in Ptolemy

Chris Shaver  Dai Bui  Stavros Tripakis
University of California, Berkeley

Motivation

- Dataflow models processing arrays of data.
- ArrayOL formalism for multidimensional dataflow.
- Coarse and fine grain parallelization.
- SpearDE, developed by Thales.

Goals

- Develop the Pthales domain for multidimensional dataflow.
- Use the Ptolemy platform as a basis for research in multidimensional dataflow models in the context of heterogenous system semantics, code generation, scheduling techniques, and the semantics of time.
- Implement the semantics of SpearDE and ArrayOL in Pthales.
- Formulate semantics for static and dynamic models.
- Integrate the Pthales domain with other models of computation.
- Develop code generation and hardware mapping.
- Experiment with temporal semantics in Pthales.

A Pthales Model

Pthales Port Semantics

At each port, Pthales actors produce and consume patterns of data tiled within arrays. Ports in the model are given a set of array data parameters of the following form:

\[
\text{parameter} = [a = x_a, b = x_b, \ldots]
\]

where \((a, b, \ldots)\) are names and \((x_a, x_b, \ldots)\) are associated data.

Illustration

Computing Parameters

A parameter set \((\vec{b}, \vec{p}, \vec{h}, \vec{t}, \vec{s}, \vec{r})\) is overdetermined with the constraint that the repetitions must fill the array as much as possible. \(\vec{s}\) can be calculated from \(\vec{r}\):

\[
\vec{s} \geq \vec{b} + \{\vec{t}, \vec{r} - 1\} + \{\vec{p}, \vec{h} - 1\} + 1
\]

where \(\langle \vec{a}, \vec{b} \rangle\) denotes point-wise multiplication.

If \(\vec{s}\) is given, \(\vec{r}\) can be calculated:

\[
\vec{r} \leq \{\vec{s} - \vec{b} - 1 - \{\vec{p}, \vec{h} - 1\} - \vec{t}, \vec{r}^{-1}\}
\]

the greatest \(\vec{r}\) satisfying this condition being:

\[
\vec{r} = \floor{\{\vec{s} - \vec{b} - 1 - \{\vec{p}, \vec{h} - 1\} - \vec{t}, \vec{r}^{-1}\}}
\]

Future Work

- Embedding Pthales semantics into Process Networks.
- Expanding specification language to include expressions as parameters.
- Automatic generation of parameters from other specifications.
- Code generation for multicore platforms.
- Pipelined scheduling techniques.
- Hierarchical characterization of dynamic semantics:
  - How can different forms of dynamic multidimensional semantics be interrelated or included in one another?
  - What is the relationship between scenario-based and modal model approaches to dynamic semantics?
Alternative Syntaxes for Ptolemy Models

Chris Shaver
University of California, Berkeley

Block Combinator Syntaxes
- Ptolemy models can be represented in a combinator syntax rather than a point-to-point syntax with named ports.
- Models can be composed, edited, or reasoned about in such a syntax.

Blocks

Operators

Parallel Composition: \( A \otimes B \)
\( \otimes : B^N_M \times B^K_P \rightarrow B^{N+K}_{M+P} \)

Serial Composition: \( A \gg B \)
\( \gg : B^N_M \times B^K_M \rightarrow B^K_M \)

Feedback Contraction: \( \langle A \rangle_n \)
\( \langle \rangle_n : B^N_M \rightarrow B^{N-n}_M \)

Permutation: \( (a_1, a_2, a_N) \)

Influences
- The Faust signal processing language.
- Diagrammatic Linear Algebra.
- Milner’s Calculus of Communicating Systems.

Abstract Syntax
Syntax of block expression (both visual and textual):
- \( E = \text{syn Name} \mid E \otimes E \mid E \gg E \mid (E)_n \)
- \( \text{Bind} = \text{syn Name} \leftarrow E \) where a Name can be a primitive block or bound to an \( E \).

Primitives
- Identity: \( id \)
- Initiator/Terminator: init, term
- Split/Merge: \( <N,> M \)
- Permutation: \( (a_1, a_2, a_N) \)

Typical Ptolemy Model

Textual Combinator Form

Visual Combinator Form

Conversion from Ptolemy to Combinator Form
- Starting with a Ptolemy model, the point-to-point syntax can be converted into a combinatorial visual syntax.
1. Split/Merge primitives replace multiply connected relations.
2. Initiator/Terminator primitives are added to unconnected ports.
3. Feedback edges are cut and drawn out to input/output pairs.
4. Organize blocks into columns dependent on predecessor columns.
5. Insert identity operators to make columns opaque.
6. Order identities to the bottom of columns.
7. Insert permutation primitives between columns.
Motivation

Ontologies can be used to analyze models. A developer defines lattices plus actor constraint rules. Here are some examples:

### Constant / Non-constant

<table>
<thead>
<tr>
<th>Nonconstant</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramp</td>
<td>Nonconstant</td>
</tr>
<tr>
<td>The output of a Ramp actor can change.</td>
<td>A Const actor’s output is constant.</td>
</tr>
</tbody>
</table>

### Observability

<table>
<thead>
<tr>
<th>Observable</th>
<th>Unobservable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>Discard</td>
</tr>
<tr>
<td>The input to a Display actor is observable.</td>
<td>A Discard actor’s input can not be observed.</td>
</tr>
</tbody>
</table>

### Abstract Interpretation (for scalar values)

- **Undefined**
- **Positive**
- **Zero**
- **Negative**
- **Unknown**

The output of a multiplication by zero is zero.

If an ontology is a lattice, the Rehof and Mogensen algorithm performs model analysis in linear time ("Tractable Constraints in Finite Semilattices", 1996). (Time is linear in the number of constraints, for finite height lattices.)

**Why compose lattices?**

A product lattice can be more powerful than a set of orthogonal individual lattices (Click and Cooper, "Combining Analyses, Combining Optimizations", 1995).

- The original rules from all sub-lattices are inherited.
- The developer writes a few new rules taking advantage of all sub-lattices.

**Current status and objectives**

The product lattice is:

- Automatically created from sub-lattices.
- Checked to ensure that the result is also a lattice.

Remaining challenges include:

- Allowing the user to edit the product lattice, especially Top and Bottom concepts.
- Some concepts in the product lattice don’t make sense. Allow users to note these with acceptance criteria.
- Combining finite and infinite width lattices.
- Combining inherited and new rules and ensuring monotonicity.
Background

Using ontologies allows us to statically analyze Ptolemy models in a principled way. The first step to doing so is to represent the domain of interest as a lattice-based ontology.

Example Ontologies

By formally expressing a body of knowledge as a lattice-based ontology, we can encode domain knowledge to inform analysis.

Infinite Ontologies

While many types of knowledge can be encoded in finite ontologies, infinite ontologies can express a larger class of properties. We have found two patterns of infinite ontologies to be broadly useful:

- **Infinite Flat Lattices**: Useful for including value information into the ontologies.
- **Recursive Lattices**: Useful for including structured information corresponding to structured data types.

Acknowledgements

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESIS) at UC Berkeley, which receives support from the National Science Foundation (NSF) awards #1020082 (CSS-EHS: PRET), #1119572 (CPS: TIDES), and #1013453 (ActionWeb); the U. S. Army Research Office (ARO) W911NF-07-1-0419; the U. S. Air Force Office of Scientific Research (MURI 140) FA9550-08-1-0112; the Air Force Research Lab (AFRL), the Multiscale Systems Center (MShy), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.
Towards flexible and robust cyber-physical systems through self-organization

Systems run in worst case mode determined at design time.

Timing constraints are given by the worst case scenario.

Sensor sampling rates and ESC control task periods run at a speed to avoid oversteering of the car at max. speed.

At best, there are different modes of operation available.

System is given no information to react in an aware manner => Timing is tested in complex HiL simulations to show that safety requirements are met.

Reasoning about timing as the key.

_Giotto_ language allows to reason about timing explicitly.

However, once the schedule is computed, it is static and not ought to be changed during run-time.

From the timing properties, such as task periods and deadlines, used for describing the timing behavior of the system it is not possible to derive the reasons for the timing constraints, the information is missing.

Recent results from Prof. Tabuada have shown that control tasks show stable behavior while _self-triggering_.

This is especially interesting because timing is derived from the physics of the system => there is a meaning, it is not just a property of the worst case analysis.

To be really able to adapt timing to changing conditions, timing information must be coupled to the goals of the system.

In the ESC example, sensor sampling rates and task periods could be adjusted to the current speed of the car.

Goal of this ongoing work: Equip run-time models with needed information to adapt their timing behavior at run-time.

In the case of control applications, stability must be guaranteed when changing timing settings.

Andreas Thuy
University of Paderborn/C-LAB
athuy@c-lab.de
1. Component-based design

Questions:
- What are the right building blocks?
- Which ones to use? How to connect them?
- What is a component? How to reason about components?

Components → systems

2. Composition

- Components: \( A, B, ... \)
- Composition operators: \( A \mid | B = C; C \circ D = E, ... \)
- Substitutability: can I replace \( C \) by \( F \) in \( E \)?

3. Abstraction–Refinement

Two faces of the same golden coin:
- \( B \sqsubseteq A \):
  - \( A \) is an abstraction of \( B \)
  - \( B \) is a refinement of \( A \)


5. This work

Deterministic abstractions for non-deterministic systems!

E.g., MP3 player:
- Deterministic specification: throughput = 44.1kHz, latency = 50 ms
- Performance preserved by stepwise refinement.

6. Example: MP3 player

7. Actors

Formal descriptions of components:
\[ A = (P, Q, R_A) \]
- \( P \): set of input ports.
- \( Q \): set of output ports.
- \( R_A \subseteq \text{Tr}(P) \times \text{Tr}(Q) \): relation between input and output event traces.

8. Event traces

Event trace: port-wise vector of event sequences.
Event sequence: sequence of “pure” (non-valued) events in time.

9. Examples of actors

Variable-delay actor \( \Delta[n_{d_i}]: \)
\[ x_{\Delta[n_{d_i}]} y \iff |x(p)| = |y(q)| \land \forall n < |x(p)|:
\]
\[ x(p)(n) + d_i \leq y(q)(n) \leq x(p)(n) + d_j \]
\[ (n > 0 \implies y(q)(n) \geq y(q)(n - 1)) \]

Finite-automaton actor:
\[ \text{Finite-automaton actor:} \]

10. The earlier-the-better refinement

\[ B = (P, Q, R_B) \] refines \( A = (P, Q, R_A) \), denoted \( B \sqsubseteq A \), iff
1. \( \text{in}_B \subseteq \text{in}_A \) (legal inputs of \( B \) are also legal in \( A \));
2. \( \forall x \in \text{in}_A \exists y \in \text{in}_B : x \text{Tr}B \implies \exists y' \in \text{in}_B : y \text{Tr}B \)

Where \( y \subseteq y' \) if events in \( y \) happen no later than those in \( y' \). E.g., \( \tau_1 \sqsubseteq \tau_2, \tau_2 \sqsubseteq \tau_1, \text{but } \tau_1 \not\sqsubseteq \tau_2 \).

Example: CSDF (cyclo-static dataflow [BELP96]) actor refining SDF actor:

11. Results

- Refinement is compositional w.r.t. parallel, serial and feedback composition (under some conditions).
- Refinement preserves worst-case throughput and latency.
- Algorithms to check refinement and compute compositions for various finite representations (SDF, automata, ...).
- Semantical unification of existing frameworks (dataflow, automata, service curves, ...).

12. References

Motivation

- Using a uniform bit-width for FPGAs is inefficient
  - Uniform bit-width selection is useful for DSP and other fixed-width systems
  - Doesn’t take advantage of the flexibility of FPGAs
  - Finding the optimal bit-width reduces area and increases clock rate while maintaining the quality of the answer
- Finding the optimal bit-width is difficult
- Many techniques exist for finding near-optimal bit-widths:
  - Statistical Simulation
  - Feed-forward Heuristics
  - SAT / ILP solver
- Ptolemy provides a good infrastructure on which to implement these algorithms

New Ptolemy Director

- Based on SDF director
- Ends when all strategies are finished
- Strategies are like sub-directors
- Director runs strategies to find range, then runs strategies to find precision
- Prints a report when all strategies have completed

New Tokens

- Implemented new Range Tokens and Simulation Tokens
- Range Tokens inherit from ScalarToken
- Allows math with ranges and constants
- Although represented by more than one number, a range token should be treated as a scalar
- Error Tokens
  - Holds two range tokens:
  - Suggested Bit-width
  - Dynamic range
  - Still represents a scalar entity

Range Algorithms

- Interval Arithmetic:
  - Simple method for calculating range:
    \[ X = [x_l, x_u], Y = [y_l, y_u], z = [x_l + y_l, x_u + y_u] \]
  - Cannot be used in feedback systems
- Affine Arithmetic:
  - Accounts for correlations between the inputs, e.g. \[ X = a + b \]
  - Interval Arithmetic: \[ X = [a - b, a + b] \]
  - Affine Arithmetic: \[ X = [a - b, a + b] \]
  - Can be used to solve for the range of FIR filters (feedback systems)

Precision Algorithms

- Most published techniques involve heuristic competitions to find a near optimal bit-width
- Competition:
  - Once the range is found, the system error can be calculated
  - The winning operator in each iteration is the one that:
    - Increases the error the least
    - Decreases the area the most
  - Competition continues until user constraint can no longer be met

Results

- Several simple test benches have been created:
  - FIR and IIR filters
  - DCT
  - RGB to YUV converter
  - BPSK timing loop
- Results for BPSK timing loop closely match those of human selected values:

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Selected</th>
<th>Accuracy</th>
<th>Best Approximated</th>
<th>Program Width</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Filter</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>NCO</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>ZCTED</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum1</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum2</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum3</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum4</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum5</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum6</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum7</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum8</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
<tr>
<td>Sum9</td>
<td>0.71</td>
<td>0.64</td>
<td>0.71</td>
<td>1.5x10^6</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Work funded by NSF Center for High-Performance Reconfigurable Computing
Introduction

Many real-time distributed systems require some shared notion of time. When measuring or acting on the physical world, the relative times of these actions between devices in a distributed system can be of significant importance. The accuracy required for this time synchronization is application specific, and can range anywhere from seconds to nanoseconds. Some areas which can utilize accurate time synchronization are test and measurement, industrial automation and control, and power generation.

The IEEE 1588-2008 standard defines a Precision Time Protocol (PTP) to synchronize time in a distributed system over a network to the sub-microsecond range. The protocol was developed to bridge the gap between the existing common methods of time synchronization, where Network Time Protocol (NTP) may not be accurate enough and Global Positioning System (GPS) may be too expensive.

IEEE 1588 Basics

Architecture

The protocol operates on a hierarchical master-slave distribution where slave clocks synchronize their time to master clocks. A best master clock (BMC) algorithm is used to select master clocks. Basically, all clocks in a network exchange information about their stability and accuracy, and the best clock is elected as the master.

Implementation

Hardware

The implementation is being developed on a Renesas 7216 Demonstration Kit with a 32-bit RISC SuperH CPU. The DP83640 Precision PHYTER from National Semiconductor is used to provide hardware support for IEEE 1588 in the form of an integrated 125MHz IEEE 1588 clock, packet detection and timestamping with 8ns precision, and synchronized event timestamping and triggering through GPIO ports.

Results

The basics of the protocol are implemented, and two directly connected boards are able to achieve synchronization within 25ns. Once connected through a router which doesn't support IEEE 1588, time synchronization is only accurate to around 100ns. The implementation was also successfully tested in a network with another implementation of the protocol. Future work involves fully supporting the protocol and tuning the servo algorithm.

Acknowledgments

The authors acknowledge the support of the Multiscale Systems Center, one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program. The authors would also like to thank Slobodan Matic and John Eidson for their guidance.
From PTIDES to PtidyOS: Programming Distributed Real-Time Embedded Systems

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET), #1035672 (CPS: PTIDES), and #0931843 (ActionWebs)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force, one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota.

Programming Model
Programming Temporally Integrated Distributed Embedded Systems
Based on Discrete-Event semantics
Event processing in time-stamp order

Clock Synchronization Simulation
Notions of Time:
- Top level simulate real world (oracle)
- Each platform contains a clock to keep track its notion of platform physical time
- Each Ptides director has a notion of model time

Simulation of platform clock Synchronization:
Slave’s platform clock tracks master’s platform clock, as shown on the output graph.

PtidyOS

- Steps away from threading model
- To ensure event of highest priority is always processed first, interrupts play an important role
- Event processing is done within interrupt service routines
- Reentrant interrupts
- No dynamic memory allocation

Ptides Simulator

- Is a library linked against application C code
- Implements Ptides semantics
- Local execution strategy includes a safe to process analysis and resource scheduling layer
- Uses a Single Stack

- Event processing is done within
- Local execution strategy includes a safe to process analysis and resource scheduling layer
- Reentrant interrupts
- Uses a Single Stack

Application: The Tunneling Ball Device

- Cyber-physical system with real-time constraints
- Event-driven signals (motor encoding, object detection)
- Parallel an automotive engine control unit (ECU)
- Timed semantics ensure:
  - Deterministic control delay
  - Data associated with the same physical time are processed together

Balls drop above a rapidly spinning disc
Device must adjust disc to allow ball to pass through a small hole, without stopping disc

WCET:
\[ t = t + d_o \]
\[ d_o = do \]

WCET:
\[ t = t + d_o \]
\[ d_o = do \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

* \( t \) is the platform physical time an event is delivered from the sensor to the rest of the platform. \( t \) is the timestamp of the sensor event.

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + \text{Sensor Delay} \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ t \leq t + d_o \]

\[ \text{Sensor Delay: } d_o = 0.5 \]

\[ T \leq t \]
The Dataflow Interchange Format: Towards Co-design of DSP-oriented Dataflow Models and Transformations

Shuvra S. Bhattacharyya
Maryland DSPCAD Research Group
http://www.ece.umd.edu/DSPCAD/home/dspcad.htm

Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies
University of Maryland, College Park, 20742, USA.


Outline

- Introduction to the dataflow interchange format (DIF) project, dataflow transformations, and DICE
- Application case study: high energy physics
- Wrapup
The Dataflow Interchange Format (DIF)

- DIF captures coarse grain dataflow applications formally [4]
- To formally describe applications, the DIF Language (TDL) is
  - Designed to capture a variety of dataflow models
  - Can be used in conjunction with functionally simulatable actor descriptions
- To facilitate design, the DIF Package (TDP) provides:
  - Scheduler, simulator, analyzers

Other benefits to beginning with a formal description:
- Bounded memory and deadlock detection
- Buffer and communication minimization:
  - Parallel, Multirate loop, or Quasi-static scheduling
- Heterogeneous task mapping and co-synthesis
- Probabilistic design, Data partitioning, Vectorization, ...

The DIF Language: Sketch

```
[dataflowModel] graphID {
    basedon {
        graphID;
    }

    [topology] {
        nodes = ndID, ...;
        edges = edgeID(srcNdID, snkNdID), ...;
    }

    [builtInAttr] {
        elementID = value;
        elementID = id;
        elementID = id1, id2, ...;
    }

    [attribute] usrDefAttr {
        elementID = value;
        elementID = id;
        elementID = id1, id2, ...;
    }

    [refinement] {
        ...
    }
}
```
Evolution of Dataflow Models of Computation for DSP: Examples

- Kahn process networks [Kahn 1974]
- Synchronous dataflow, [Lee 1987]
  - Static multirate behavior
  - SPW (Cadence), National Instruments LabVIEW, and others.
- Well behaved stream flow graphs [1992]
  - Schemas for bounded dynamics
- Boolean/integer dataflow [Buck 1994]
  - Turing complete models
- Multidimensional synchronous dataflow [Lee 1992]
  - Image and video processing
- Scalable synchronous dataflow [Ritz 1993]
  - Block processing
  - COSSAP (Synopsys)
- CAL [Eker 2003]
  - Actor-based dataflow language
- Cyclo-static dataflow [Bilsen 1996]
  - Phased behavior
  - Eonic Virtuoso Synchro, Synopsys El Greco and Cocentric, Angeles System Canvas
- Bounded dynamic dataflow
  - Bounded dynamic data transfer [Pankert 1994]
- The processing graph method [Stevens, 1997]
  - Reconfigurable dynamic dataflow
  - U. S. Naval Research Lab, MCCI Autocoding Toolset
- Stream-based functions [Kienhuis 2001]
- Parameterized dataflow [Bhattacharya 2001]
  - Reconfigurable static dataflow
  - Meta-modeling for more general dataflow graph reconfiguration
- Reactive process networks [Geilen 2004]
- Blocked dataflow [Ko 2005]
  - Image and video through parameterized processing
- Windowed synchronous dataflow [Keinert 2006]
- Parameterized stream-based functions [Nikolov 2008]
- Enable invoke dataflow [Plishker 2008]
- Variable rate dataflow [Wiggers 2008]

DIF Project Components

- Core components
  - The DIF language (TDL)
  - The DIF package (TDP)
  - Enable invoke dataflow (EIDF) and functional DIF
  - DIFML: XML dialect
- Plug-ins
  - DIF-to-C: Software synthesis for SDF
  - TDIF and TDIFSyn
  - The dataflow schedule graph (DSG)
- Interfaces to ADS, OpenDF, LabVIEW, Ptolemy II, …
High Level Dataflow Transformations

- A well designed dataflow representation exposes opportunities for high level algorithm and architecture transformations.
- High level of abstraction → high implementation impact
- Dataflow representation is suitable both for behavior-level modeling, structural modeling, and mixed behavior-structure modeling
  - Transformations can be applied to all three types of representations to focus subsequent steps of the design flow on more favorable solutions
- Complementary to advances in
  - C compiler technology (intra-actor functionality)
  - Object oriented methods (library management, application service management)
  - HDL synthesis (intra-actor functionality)

Representative Dataflow Analyses and Optimizations

- Bounded memory and deadlock detection: consistency
- Buffer minimization: minimize communication cost
- Multirate loop scheduling: optimize code/data trade-off
- Parallel scheduling and pipeline configuration
- Heterogeneous task mapping and co-synthesis
- Quasi-static scheduling: minimize run-time overhead
- Probabilistic design: adapt system resources and exploit slack
- Data partitioning: exploit parallel data memories
- Vectorization: improve context switching, pipelining
- Synchronization optimization: self-timed implementation
- Clustering of actors into atomic scheduling units
Formal Model Detection (Core Functional Dataflow [3])

- Divide actors into a set of modes
  - Each mode has a fixed consumption and production behavior
- Write the enabling conditions for each mode
- Write the computation associated with each mode
  - Including next mode to enable and then invoke
- For example, consider a standard Switch:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Consumes</th>
<th>Produces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Control</td>
<td>Data</td>
</tr>
<tr>
<td>Control</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>True</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>False</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Switch Actor

Practical Model Detection on Units

- Deterministic – Does the output repeat?
  - Input Sequence 1: 00101110
  - Input Sequence 2: 00101110
  - Output Sequence 1: 1101101000
  - Output Sequence 2: 1101101000

- Statefulness – Does the output just reorder?
  - Input Sequence 1: 00101110
  - Input Sequence 2: 00101011
  - Output Sequence 1: 1101101000
  - Output Sequence 2: 1100001101

- Dataflow model – Does input & output behavior repeat?
DICE: DSPCAD Integrative Command-Line Environment [2]

What it is…
• a framework for managing cross-platform testing
• language independent
• an open source resource

What it does not do
• provide code synthesis or debugging tools
• provide simulation capabilities
• transcode between platforms or languages

Outline

• Introduction to the dataflow interchange format (DIF) project, dataflow transformations, and DICE
• → Application case study: high energy physics
• Wrapup
Case Study: Compact Muon Solenoid Trigger

- **Complex:**
  - 9300 magnets
  - Protons travel at 99.99% times the speed of light
  - 7 TeV beam collisions

- **Performance Oriented:**
  - 6 collision detectors
  - 600 million proton collisions per second

- **International Collaboration:**
  - 2000 Scientists
  - 155 Institutes
  - 37 Countries

CMS Trigger Background

- **Large Hadron Collider (LHC):**
  - CERN: Swizerland/France
  - Event rate of 1GHz
  - Trigger Selectivity: ratio of trigger rate to event rate (e.g., $10^{-11}$)

- **Compact Muon Solenoid**
  - General purpose particle physics detector for the LHC
  - CMS Trigger: Multi-Level Filtering: Level 1 (FPGA) → High Level Trigger (software) → Tape storage

Goals: Efficient, Agile Design

• The upgraded Calorimeter Trigger will require new algorithms

• Modern field programmable gate arrays (FPGAs) provide efficient platforms

• Implement Calorimeter Trigger using
  – A unified design platform
  – Unified design and test methodologies
  – Techniques that facilitate future upgrades

• Start by implementing a baseline design for the new algorithms

Solution: Novel Implementations and a Unified Cross-Platform Management System

• Collaboration with University of Wisconsin [1]

• Novel FPGA designs
  – Reexamination of physics algorithms for FPGAs
  – Structured analysis of resource usage

• Cross-platform design management
  – Novel, light weight development framework
  – Cross-platform unit testing
  – Dataflow model detection
  – Enhanced auto-documentation
Impact: Performance and Cost

- Novel FPGA implementations for over a dozen modules in the CMS detector
  - Improve performance
  - Cut implementation costs by reducing the number of FPGAs required for the upgrade
- New design process
  - Bugs found earlier in design process saves time and money
  - Automated documentation facilitates fast collaborative design process

Processing Detectors

- 56x72 sized grids
- With millions of events a second, storing all of the data would result in GigaBytes per second
  - Instead, store only events that trigger certain conditions
  - L1 trigger finds image features that represent certain particles from a series of:
    - Thresholding
    - Filtering
    - Sorting
  - Must complete in nanoseconds to process every sample period
Triggering Application Graph

Written by application designers and then re-implemented by hardware engineers → Cross-platform verification is a problem
Test directory structure

Model based testbench creation

Text files (sample input provided by user)
Results

<table>
<thead>
<tr>
<th>Actor</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Deterministic</th>
<th>Model Detected</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Thresh</td>
<td>12</td>
<td>12</td>
<td>Yes</td>
<td>HSDF</td>
<td>No</td>
</tr>
<tr>
<td>Cluster Compute</td>
<td>12</td>
<td>6</td>
<td>Yes</td>
<td>HSDF</td>
<td>No</td>
</tr>
<tr>
<td>Overlap Filter</td>
<td>8</td>
<td>4</td>
<td>Yes</td>
<td>SDF</td>
<td>No</td>
</tr>
<tr>
<td>Jet Reconstruction</td>
<td>1</td>
<td>2</td>
<td>Yes</td>
<td>SDF</td>
<td>No</td>
</tr>
</tbody>
</table>

\[(H)SDF = (homogeneous) synchronous dataflow\]

Outline

- Introduction to the dataflow interchange format (DIF) project, dataflow transformations, and DICE
- Application case study: high energy physics
- → Wrapup
The dataflow interchange format (DIF) project
- The DIF Language (TDL)
- The DIF Package (TDP)
- Plug-ins for simulation and synthesis

The DSPCAD Integrative Command Line Environment (DICE)

Application case study: high-energy physics

Other ongoing application thrusts in the DIF project include:
- embedded speech processing
- software-defined radio
- wireless sensor networks
- image registration
- radio astronomy instrumentation

Co-design of dataflow-based representations and transformations

Summary

Portions of the work presented here have been sponsored by DARPA (through MCCI), and NSF (ECCS0823989 and CNS0720596).

For more details on these projects, and associated publications:
http://www.ece.umd.edu/DSPCAD/home/dspcad.htm.
To Probe Further …

(Available from: http://www.ece.umd.edu/DSPCAD/papers/contents.html)


Workflow Fault Tolerance for Kepler

Sven Köhler, Timothy McPhillips, Sean Riddle, Daniel Zinn, Bertram Ludäscher

Introduction

- **Scientific Workflows**
  - Automate scientific pipelines
  - Have long running computations
  - Often contain stateful actors
- **Workflow execution can crash because of …**
  - Hardware failures
  - Power outages
  - Buggy / malicious actors, …
- **Current approach**: Start workflow from the beginning
Current Fault Tolerance Solutions ...

- Manage actor failures or sub-workflow failures AND their effects
  - Atomicity and provenance support for pipelined scientific workflows [Wang et al.]
  - Ptolemy’s “Backtrack” [Feng et al.]

- Use caching strategies for faster re-execution
  - W.A.T.E.R.S. memoization [Hartman et al.]
  - “Skip over” strategy [Podhorszki et al.] (CPES)

Our Fault Tolerance Approach

- Recovery based on readily available Provenance
  1. Create a uniform model for workflow descriptions and provenance
  2. Record actor state in provenance in relation to invocations
  3. After a workflow crash: Use provenance data in our uniform model and start recovery

- Different strategies for recovery
### Our Recovery Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>- Restart the workflow without using provenance</td>
</tr>
<tr>
<td></td>
<td>- Re-executes everything</td>
</tr>
<tr>
<td>Replay</td>
<td>- Use basic provenance to speed up recovery</td>
</tr>
<tr>
<td></td>
<td>- Re-execute stateful actor with input from provenance (replay)</td>
</tr>
<tr>
<td></td>
<td>- Restore all queues</td>
</tr>
<tr>
<td></td>
<td>- Resume the workflow according to the model of computation</td>
</tr>
<tr>
<td>Checkpoint</td>
<td>- extension of replay strategy</td>
</tr>
<tr>
<td></td>
<td>- Use checkpoints (state of actors stored in provenance)</td>
</tr>
<tr>
<td></td>
<td>- Reset stateful actors to recorded state</td>
</tr>
<tr>
<td></td>
<td>- Replay successful invocations after the checkpoint</td>
</tr>
<tr>
<td></td>
<td>- Restore queue content</td>
</tr>
<tr>
<td></td>
<td>- Resume the workflow</td>
</tr>
</tbody>
</table>
Example: Checkpoint in SDF

Workflow with a mix of stateful and stateless actors

Corresponding schedule of the workflow with a fault during invocation B:2

Execution with a Failure

Execution of the previous workflow

Checkpoints for actor B and D but not for C

At invocation B:2 - Crash

Tokens t4 and t7 - in queue

Token t9 - to be restored

Token t10 - to be deleted
Using Kepler with the Provenance Recorder

Extensions to the Provenance Recorder:
- Record serialized tokens
- Extend the provenance schema
- Add queries

Recovery Extension in the SDF Director:
- Serialize states after one iteration of the SDF schedule
- Black-list to prevent capturing transient actor information
- White-list if actors are annotated with state-information
Prototype Implementation in Kepler

- **Upon restart:**
  - SDF director checks provenance information
  - SDF director calls the recovery engine

- **Recovery:**
  - Restore the internal state of actors
  - Replay successful invocations using input tokens from provenance
  - Restore content of all queues
  - Return to SDF director with information about where to resume

---

Evaluation

Synthetic Workflow | Results

<table>
<thead>
<tr>
<th>SDF Director</th>
<th>A (stateful)</th>
<th>B (stateless)</th>
<th>C (stateful)</th>
<th>D (stateless)</th>
<th>E (stateful)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15 sec</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph showing synthetic workflow results](image)

- Completion
- Recovery
- Time to Crashpoint
Advantages of our strategy:
- Efficient workflow recovery using readily available information
- Quick constant time recovery (checkpoint strategy)
- Generalized approach, saving labor
- Robustness

Disadvantages of previous strategies:
- Required labor-intensive customized systems
- Failure required restarting long-running workflows from the beginning
- Caching only works for stateless actors
- Caching only provides a partial recovery
Design, Analysis, and Implementation of Static Dataflow Models for Hardware Targets,

Abstract: We present the DSP Designer framework to implement applications specified in the Static Dataflow (SDF) model of computation on hardware targets, such as FPGAs. Prior studies have shown the effectiveness of SDF as a natural model to specify multi-rate streaming applications. However, the focus of these works has primarily been on SDF implementations for processor targets. DSP Designer specializes the SDF model to make it suitable for hardware targets. It facilitates hardware actor definition and intellectual property (IP) integration. The back end additionally provides analysis methods tuned for synthesis of efficient hardware designs, such as resource allocation, memory optimization, and scheduler generation. The objective is to deliver an exploration framework that empowers application domain experts to become hardware designers. In this talk, we highlight key concepts underlying DSP Designer, demonstrate preliminary capabilities for exploration and implementation using practical applications, and discuss open challenges related to the specification of control and timing along with dataflow. We also summarize key features of the DSP Designer software architecture and invite partners to leverage our infrastructure and API to build tools for graphical design.
**Kepler/G-Pack: A Kepler Package Using the Google Cloud for Interactive Scientific Workflows**

*Gongjing Cao, Lei Dou, Quinn Hart, Bertram Ludaescher*

*UC Davis*

---

### Interactive Scientific Workflows

- **Requirements for human interaction in scientific applications**
  - dynamic branching based on scientists’ runtime decision
  - semi-automatic data curation
- **Category of human interaction**

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>time cost</td>
<td>short time, instantly</td>
<td>unknown</td>
</tr>
<tr>
<td>people involved in interaction</td>
<td>workflow executor</td>
<td>people other than workflow executor</td>
</tr>
<tr>
<td>workflow blocking</td>
<td>yes</td>
<td>not necessarily</td>
</tr>
<tr>
<td>implementation approaches</td>
<td>graphical window</td>
<td>dedicated server</td>
</tr>
<tr>
<td></td>
<td>web page/browser</td>
<td>mail, polling, callback</td>
</tr>
</tbody>
</table>

---

*Ptolemy Miniconference, February 16, 2011  DAKS, UC Davis*
Google Cloud Computing in Kepler

- Authorization
  - 1st step to acquire access to Google services
- Spreadsheet Operations
  - Various manipulations on Google Spreadsheet, like copy, share, import, export, query, audit.
- Data Analysis
  - Various operations especially for data curation purpose, like duplicates identification and fuse, data boundary inspection.
- Data Access
  - Google visualization datasource actor allows SQL-like access to Google cloud data
- Mail Service
  - MailSender Actor supports sending email through SMTP with UserName/Password or OAuth token/secret.
OAuthAuthorizer Actor

1. Requests token with OAuth
2. Responds with unauthorized request token
3. Invokes Web browser with request URL
4. Requests authorization for request token
5. Redirects with verification token
6. Feeds verification token to installed app
7. Requests exchange for OAuth token & secret
8. Responds with OAuth token & secret
9. Requests data with OAuth token & secret
10. Responds with requested data

Spreadsheet Operation Actors

<table>
<thead>
<tr>
<th>Actors</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importer</td>
<td>import data to a spreadsheet</td>
</tr>
<tr>
<td>Exporter</td>
<td>export data from a spreadsheet</td>
</tr>
<tr>
<td>Copy</td>
<td>copy a spreadsheet from a template</td>
</tr>
<tr>
<td>Share</td>
<td>share the spreadsheet with another user</td>
</tr>
<tr>
<td>Query</td>
<td>query data from the spreadsheet</td>
</tr>
<tr>
<td>Auditor</td>
<td>allow human interaction during the execution of the workflow</td>
</tr>
<tr>
<td>PollingQuery</td>
<td></td>
</tr>
</tbody>
</table>
Data Access
VisualizationDataSource Actor

- Get the response from a servlet which is implemented with Google visualization data source.
- e.g. servlet:
  - http://comet.cs.ucdavis.edu:8080/CimisVis/sql

Table: daily
query: select * where d_date>date'2005-01-09' and d_date<date'2005-01-20'

Data Access
VisualizationDataSource Actor

- Actor parses JSON string to Kepler token
- array of RecordToken
MailSender Actor

- Gmail and other mail server supporting SMTP protocol
  - It supports sending email through SMTP with UserName/Password or OAuth token/secret.

Data Analysis Actors (COMAD actors)

<table>
<thead>
<tr>
<th>Actors</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering</td>
<td>Do clustering on a list of RecordToken by applying specified function for specified field.</td>
</tr>
<tr>
<td>DataFuser</td>
<td>Fuse a list of RecordToken with specified function</td>
</tr>
<tr>
<td>ConditionTester</td>
<td>Test whether specified condition is satisfied or not.</td>
</tr>
</tbody>
</table>
Evapotranspiration Workflow

Spreadsheet is used as data storage and calculation tool during this demo. ETOs are calculated from CIMIS weather station data according to different models.

Biofuel Refinery Workflow

This workflow allows users to copy a spreadsheet from a template, then use the existing data or edit the spreadsheet to help them make decisions in building a biofuel refinery.
SpecimenRecordMerge Workflow

This workflow demonstrates how specimen data with duplicates are fused and curated semi-automatically.

1. Firstly the specimen records are imported from a file in csv format.
2. Secondly the duplicated sets of specimen records are identified through specific clustering function and for each duplicate set a fused record is generated.
3. Thirdly the original specimen records and the fused record are imported into a google spreadsheet for human curation. Meanwhile the corresponding curators are informed of such curation request.
4. Finally, the human curation result is collected once it’s finished by the curator.

AdvancedSpecimenRecordMerge Workflow

This workflow makes the following improvement of SpecimenRecordMerge workflow. It’s demonstrated how easy it is to reconfigure COMAD workflow to adapt to new functionalities.

1. Insert SpecimenRecCleaner actor to clean source data by removing “bad” specimen records with unclear collector of “et al.”
2. Reconfigure Clustering actor to cluster specimen records against collector field with fuzzy match method. Therefore the specimen records collected at the same time and the same location but by different collector of "E. L. Morris" and "E. Morris" could be identified as duplicates.
Thanks & Question
Context-Aware Actors

Anne H. H. Ngu
Department of Computer Science
Texas State University-San Marcos

Outline

- Why Context-Aware Actor?
- Context-Aware Scientific Workflow System
  - Architecture
  - Context Modelling and Provisioning
  - Context Providers and agents
  - Context Annotation
  - Demo of Context-Aware actors
  - Conclusion and future work
Why Context-Aware Actors?

- **Actor-oriented scientific workflow model** - ideal for modeling data intensive, stream-based and concurrent execution nature of scientific workflow.

- Problems with using Actor-oriented model for modeling dynamic workflows.
  - too many low-level control-flow actors and wiring lead to complex workflow that is hard to comprehend and re-use.

- Proposed solution in the past: Static Frame and Template, Dynamic frame actor, and Generic actor.
  - Static frame and template cannot adapt to runtime conditions (SciFlow 2006).
  - Dynamic frame encodes control-flow implementation of runtime conditions in the frame actor, thus cannot adapt to new situation without re-implementation (SSDBMS 2009).
  - Generic actor encodes all variations in control-flow logic a-priori in the actor.
  - All involve knowing low-level actor programming skills.

- **Context-aware actor** is proposed to enable actor-oriented scientific workflow to be more personalized, adaptive, intuitive, and intelligent by modeling the logic related to quality runtime adaptations as contexts and provisioned by a separate computing unit to the actor.

Context-Aware Actors

- **Context awareness**: the capability of being aware of its physical environment or situation (context) and responding proactively and intelligently based on such awareness

- One of the most important trends in computing that is becoming more important with relentless growth in mobile services (location-based services)

- **Context-aware Actors (CAA)**: make actors more personalized, adaptive, and intelligent

- **Actor**: a reusable component that may encapsulate external computation programs, grid services, scripts or local applications.
Context-Aware FileCopier Actor
An Example CAA

User enters source file, destination file, source machine, destination machine, and a high-level file transfer option (e.g., fast protocol)

Recommend fastest type of protocol to transfer file taking user’s contextual information such as size of the source file, availability of protocols and speed of the network connection between machines.

Example of contexts used for deciding on a protocol are:
- If filesize is > 6GB and network connection speed is < 6ms, use bbcp protocol.
- If bbcp protocol is not available and recursive option is set, use scp protocol

Context and Context-Awareness

- **Context:**
  - All data that can be gathered automatically at runtime which can affect the actor’s behavior:
    - System parameters (OS, CPU usage, job queue status, network speed, type of machines, availability of resources and their versions, occurrences of certain events)
  - All data supplied by the user (gathered manually) especially those data related to his/her preferences. This can include strategies that can be applied to obtain specific level of guaranteed quality

- **Context-Awareness**
  - The mechanism for adapting the execution of an actor based on the sensed /gathered contextual information.
Kepler Context-Aware Scientific Workflow Architecture

Context Providers
- network speed
- protocol availability
- filesize

Kepler System

Agent-1

Agent-n

Context Provisioner

Cached context

context binding

Actor 1

Actor n

Context-awareness

Execution Engine

Context Provisioning Server

Define Atomic Context
Atomic contexts are low-level contexts that do not rely on other contexts.

Define Composite Context
Composite contexts are high-level contexts that derive from one or more contexts.

Define Context Community
Context Community is a container where multiple context services are aggregated and a unified interface is offered.

Register Agent
Agent is responsible to get the context from the context provider.
Context Provisioning Server (CP)

- Is adapted from the ContextServ project whose goal is to provide a platform for rapid development of context-aware web services. It is funded by Australian Research Council.

- CP is responsible for setting up, acquiring, and managing the contexts used by any context-aware actors.

- CP contains a set of agents and a repository of context XML files (context specification)

- The context XML files store the specification of all contexts used by a specific project. CP uses the context XML files to figure out how to acquire defined context information from the appropriate agents.

- CP is implemented as a web service using Apache CXF (a light-weight Java based web services development tool kits)

Context Providers

- **Context providers** are the context sources. They are independent from CP.

- There are many different kinds of context providers (hardware sensors, PDA, software systems, web services).

- **Agents** are used to provide a uniform abstraction for obtaining context from a particular kind of context provider (e.g. Webservice agent is used to acquire contexts from all context providers which can be queried through SOAP protocol)

- Context providers can be added and removed anytime.

- It is the responsibility of the context provider to provide the implementation of services (i.e. gathering of the contextual data).
An example of a context provider

Context Provider

- Name: verifyFileSize
- Category: Remote
- Link: contextProviders.verifyFileSizeContextProvider
- Agent: CommandLineAgent

Operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>For</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSize</td>
<td>Parameter</td>
<td>target, directory</td>
</tr>
</tbody>
</table>

Implementation of verifyFileSize context provider

Use command like `ls -al filename` to check size of file

Context Modelling

- Context Provisioner distinguishes between atomic contexts and composite contexts
  - Atomic contexts: low-level contexts, directly provided by context providers
    - e.g., filesize, temperature,
  - Composite contexts: high-level contexts, no direct providers, aggregate multiple atomic or composite contexts
    - e.g., fastProtocol
    - Provide more powerful context modelling mechanism
- Context can be provided by one or more context providers
Example of Atomic Context

XML representation of atomic context

```
<context>
  <name>filesize</name>
  <type>float</type>
  <category>Atomic</category>
  <context_provider>
    <name>verifyFileSize</name>
    <category>Remote</category>
    <agent>commandLineAgent</agent>
    <link>contextProviders.FileSizeContextProvider</link>
    <operation>getFileSize</operation>
    <input ref="Parameter">target</input>
    <input ref="Parameter">directory</input>
  </context_provider>
</context>
```
Rules for recommending a fast protocol defined as a composite context

Fast protocol:

If ( (bbcp is available)

// big file, slow network speed, so better off with a fast protocol
if (filesize is >=6 GB) & (network speed >=6ms) ) then choose bbcp
else if (filesize < 6 GB OR network speed < 6 ms )
    if recursive transfer
        choose scp
    else
        if (sftp available) choose sftp
        else choose scp
else
    if recursive transfer
        choose scp
    else
        if (sftp available) choose sftp
        else choose scp

02/8/11  Ngu-TxState
Representation of composite context

<context>
  <name>fastprotocolB</name>
  <type>String</type>
  <category>Composite</category>
  <dependence>filesize</dependence>
  <dependence>netspeed</dependence>
  <dependence>bbcprotocolB</dependence>
  <dependence>srmLiteprotocolB</dependence>
  <dependence>sfprotocolB</dependence>
  <scxml>fastprotocolB.scxml</scxml>
</context>

Development of Context-Aware Actor
Context Awareness Modeling

- Context binding is the ability for an actor to bind to/access relevant contexts defined in the CP.
- Context awareness is the mechanism by which an actor behavior can be modified based on the gathered contextual information.
- A context-aware scientific workflow system must implement a context binding mechanism and a context-triggering mechanism.

Context Annotation as the Awareness Mechanism

- The main purpose of context annotation is to facilitate an end user (scientist) to tailor an existing actor in the KEPLER repository to his/her environmental context by interacting with a visual annotation tool. Context is injected into an existing workflow on demand basis.
- The end user must identify context sensitive parameters (CAObject) in an actor and bind them to appropriate contexts. We assume that the end user understand the semantics of the defined contexts in his/her domain.
- Two new classes, ContextAnnotationGUI and ContextParameter are introduced. Context annotation encapsulates all user interactions with the context provisioning server that will result in correct context binding.
- ContextParameter class encapsulates actions associated with contacting the context provisioning system to provide real-time evaluation of the contextual value of a context sensitive port or parameter in an actor.
- Context Annotation provides a generic mechanism for incorporating context without any code changes.
Implementation of Context Annotation

[Diagram showing inheritance and method definitions for FigureAction, ContextAnnotationMenu, ContextAnnotationGUI, AtomicActor, CompositeActor, TypedAtomicActor, CompositeActor, ContextParameters, ContextParameter, StringParameter.]

Implementation of ContextParameter

[Code snippet:]

```
public String getValue(Object[][] paras) throws IllegalActionException {
    // contacts the Context Provisioning server
    // saves the list of parameters from an actor
    // uses getContext() API to evaluate the context
```
Steps in Context Annotation

1. From the GUI, display all the existing parameters or ports of the actor that are candidates for context annotation.
2. Let the user select the context parameter for annotation.
3. Contacts the CP server to find the list of contexts that is relevant to the selected context parameter.
4. Let the user pick the required context to bind to.
5. If the binding requires mapping, map the context variables with actor parameters.
6. Click “map” to finish the binding and repeat 2 to 5 for the next parameter.
7. The context triggering phase started when user pressed the done button and run the actor.
8. The context computation takes place during the firing of the actor.
Summary of Context Annotation

- Context annotation presented a new framework that enabled scientific tasks to exploit dynamic environmental information during runtime without introducing complex control-flows and additional proliferate actors.

- Context annotation enables different kepler’s actors to be context-aware without any low-level re-coding of the actors.

- Context annotation simplifies the construction of scientific workflows that involve intricate adaptive behaviour, especially when a myriad of environmental conditions must be checked and verified. When environment conditions must be checked in different workflows, it can be outsourced to the CP rather than coding those conditions in every workflow.

- Intelligent defaults set up in contexts can be re-used across different workflows. This simply the actor programming.

- Context annotation provides the ability for the end user to plug in different contexts for the different situations under which a workflow can be executed. For example, the ability to choose the machine with maximum CPU, the network with shortest delay, or the protocol with highest reliability transparently for the users. It hides the complexity of running the workflow in different execution environments.

02/8/2011 Ngu-TxState

Future Work

- Apply context-aware model to Cyber Physical System
  - power grids, medical device systems, traffic control systems, environmental monitoring all invariably must deal with real time changing physical conditions. Can a more robust and adaptive CPS system be built with context-aware model?

- Integrate context-aware model with scientific workflow template design workbench.
  - Workflow templates encapsulate both control flow and data flow patterns that can be reused and adapted by scientists with minimal configuration in their pursue in designing and executing their own scientific processes. Actor and parameter bindings in template can benefit from the additional contextual information.
  - Context-driven binding of templates will result in a configured workflow that is of better quality.

02/8/2011 Ngu-TxState
Modular Synchronous Dataflow Code Generation

Dai Bui, Stavros Tripakis, Marc Geilen, Bert Rodiers, Edward A. Lee
UC Berkeley

Ptolemy Mini-Conference, February 16, 2011

Modular Code Generation

- Code generated for a composite block independently from used contexts:
Motivations

• Reusability
  • Incremental compilation
  • IP protection
• Modularity
  • Unit verification and testing
  • Parallelization
  • Scalability
• Reduction of runtime overhead
  • Speeding up simulations

Naïve SDF Code Generation

Finite-buffer schedule:FAABG

No finite-buffer schedule!
Naïve SDF Code Generation

- Initial tokens on the channel

Schedule: AAABB

Schedule: AABQAB

Deadlocked!

Modular SDF Code Generation

- Non-monolithic firing function
- DSSF = Deterministic SDF with Shared FIFOs

Non-monolithic

How do we synthesize?

Can we do (AAB)(Q)(AB)?
Synthesis

- Unfolding (non-homogeneous to homogeneous)

- Clustering

Clustering Algorithm Effectiveness

Valid clustering yet bad!

Better clustering

<table>
<thead>
<tr>
<th>Test cases</th>
<th># ins/outs</th>
<th># actors</th>
<th># nodes in unfolding graph</th>
<th># clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>2/2</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Entropy</td>
<td>1/1</td>
<td>15</td>
<td>1545</td>
<td>1</td>
</tr>
<tr>
<td>CD to DAT</td>
<td>1/1</td>
<td>4</td>
<td>156</td>
<td>15</td>
</tr>
</tbody>
</table>
Conclusions

- Hierarchical SDF models are not compositional
- Introduce DSSF profiles as a compositional representation of composite actors and show how this representation can be used for modular code generation
- Propose a synthesis algorithm that can handle hierarchical models of arbitrary depth

Future Work

- Implement more advanced clustering algorithms to reduce the number of clusters
- Understand the relations between the number of clusters, the size of generated code and performance
- Estimate throughput and delay of a code-generated model
References


Thank you
The Ptolemy Project
Advancing System Design

Edward A. Lee
Robert S. Pepper Distinguished Professor

Ninth Biennial Ptolemy Miniconference
February 16, 2011
Berkeley, CA, USA

Cyber-Physical Systems (CPS):
Orchestrating networked computational resources with physical systems

Military systems:
E-Corner, Siemens
Daimler-Chrysler

Transportation
(Air traffic control at SFO)

Power generation and distribution

Avionics

Building Systems

Telecommunications

Factory automation

Instrumentation
(Soleil Synchrotron)

Courtesy of Kuka Robotics Corp.
Courtesy of General Electric
Courtesy of GE

Courtesy of Daimler-Chrysler

Courtesy of Siemens
CPS Example – Printing Press

- **High-speed, high precision**
  - Speed: 1 inch/ms
  - Precision: 0.01 inch
  -> Time accuracy: 10us
- **Open standards (Ethernet)**
  - Synchronous, Time-Triggered
  - IEEE 1588 time-sync protocol
- **Application aspects**
  - local (control)
  - distributed (coordination)
  - global (modes)

Where CPS Differs from the traditional embedded software problem:

- **The traditional embedded software problem:**
  Embedded software is software on small computers. The technical problem is one of optimization (coping with limited resources).

- **The CPS problem:**
  Computation and networking integrated with physical processes. The technical problem is managing dynamics, time, and concurrency in networked computational + physical systems.
Approaching the CPS Challenge

Physicalizing the Cyber (PtC): to endow software and network components with abstractions and interfaces that represent their dynamics in time.

Cyberizing the Physical (CtP): to endow physical subsystems with cyber-like abstractions and interfaces

Projects at Berkeley focused on Physicalizing the Cyber

Time and concurrency in the core abstractions:

- **Foundations**: Timed computational semantics.
- **Bottom up**: Make timing repeatable.
- **Top down**: Timed, concurrent components.
- **Holistic**: Model engineering.
ON TO 2013! AND BEYOND!

Photo by Chamberlain Fong